**Assignment3**

**NUID: 001569774**

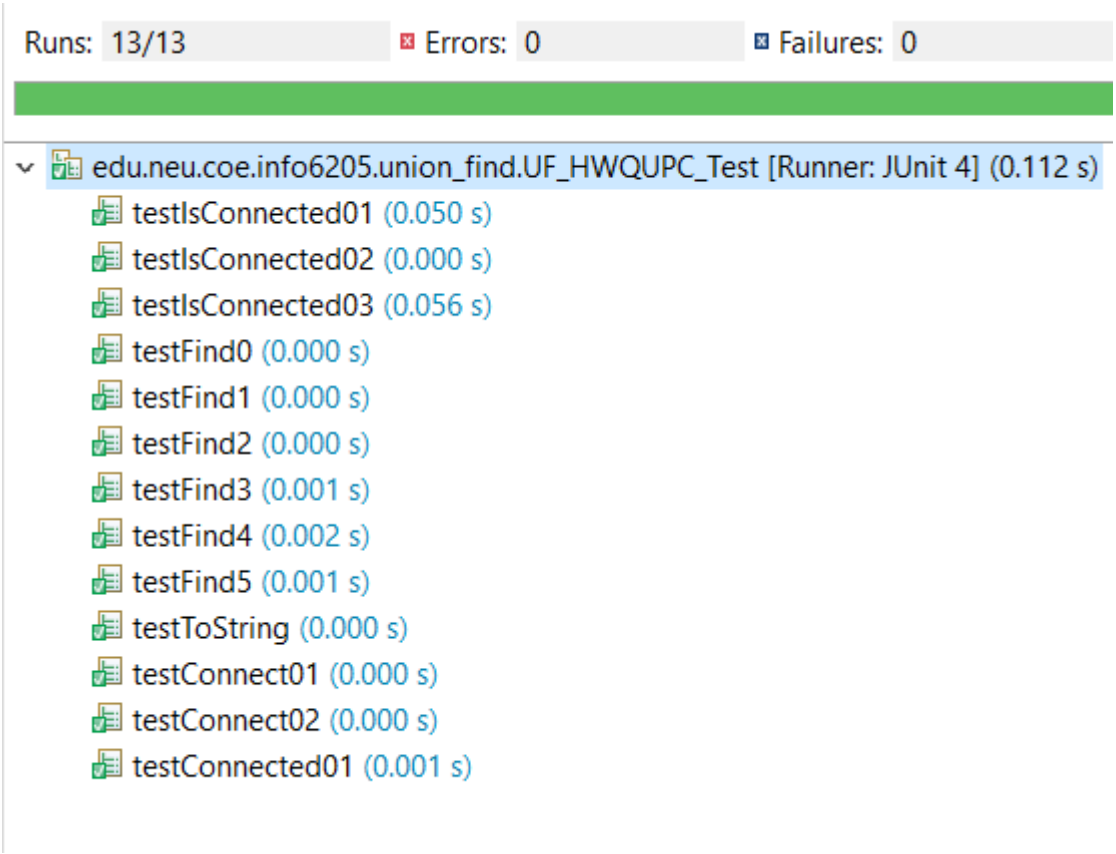
**Naga Sai Charan Guttikonda_Section5**

**Step1:** (a) Implement height-weighted Quick Union with Path Compression. For this, you will flesh out the class UF_HWQUPC. All you have to do is to fill in the sections marked with // TO BE IMPLEMENTED ... // ...END IMPLEMENTATION.

(b) Check that the unit tests for this class all work. You must show "green" test results in your submission (screenshot is OK).

(a) Code loaded into the repository.

(b): Testcase results



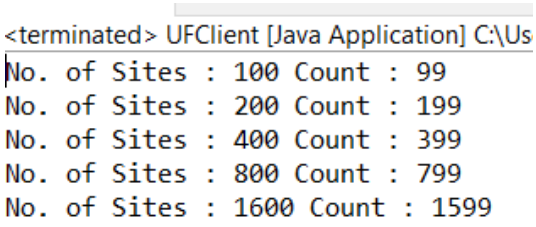
**Step2:** Using your implementation of UF_HWQUPC, develop a UF ("union-find") client that takes an integer value n from the command line to determine the number of "sites." Then generates random pairs of integers between 0 and n-1, calling connected() to determine if they are connected and union() if not. Loop until all sites are connected then print the number of connections generated. Package your program as a static method count() that takes n as the argument and returns the number of connections; and a main() that takes n from the command line, calls count() and prints the returned value. If you prefer, you can create a main program that doesn't require any input and runs the experiment for a fixed set of n values. Show evidence of your run(s).

Code loaded to the repository.

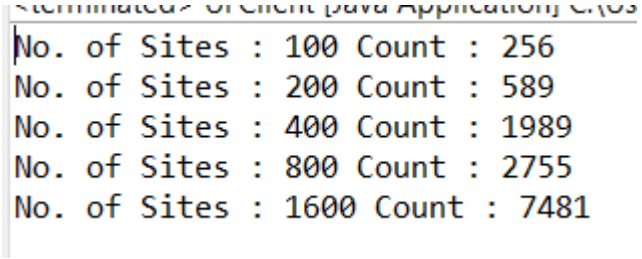
The UFClient in the repository calls the connected and union if not connected till the components function returns the value 1.

The statis count will be incremented each time a we call the function.

- ➔ When we call the function with the union, With this experiment we will know the number of connections made will be m = n – 1 ( m no of connections, n as no of pairs).




- ➔ Now let us the call the static count for the number of random pairs generated to get the components return value 1.




Where count is the number of random pairs generated to connect all the sites.

**Step3:** Determine the relationship between the number of objects (*n*) and the number of pairs (*m*) generated to accomplish this (i.e. to reduce the number of components from *n* to 1). Justify your conclusion.

Relation between no of sites as N, and no of pairs generated as M.

**M ~ 1/2*N*LogN**

To determine the relation between the no of random pairs generated and the no of sites taken.

Let us consider the random number generation method

We will consider the mean number of random pairs generated to connect all the components.

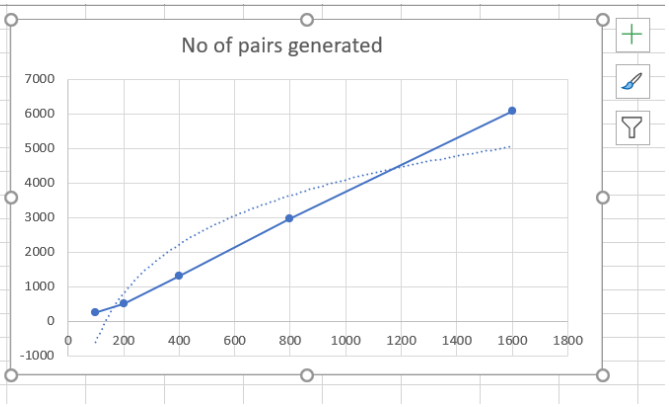
Code loaded in the repository (UFClient_3)

In this we will take each number of sites and run it 10 times and take the mean no of random pairs generated for each site doubling the no of sites.

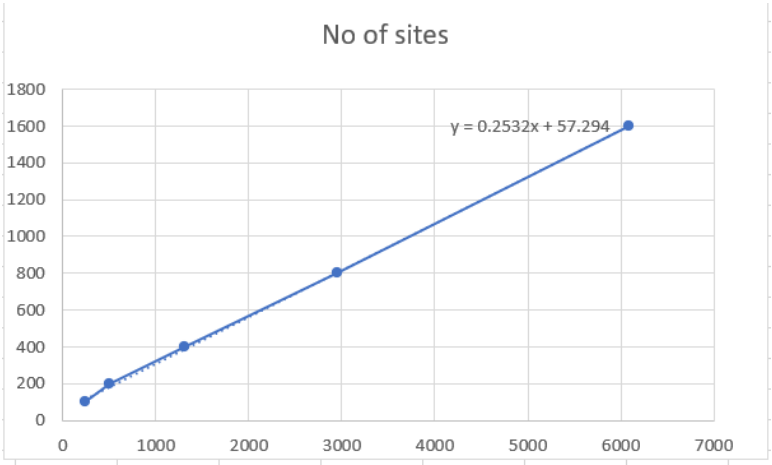
```
<terminated> UfClient_3 [Java Application] C:\Users\chara\.p2\pool\plu
No of sites: 100 mean value for 10 runs: 233
No of sites: 200 mean value for 10 runs: 556
No of sites: 400 mean value for 10 runs: 1318
No of sites: 800 mean value for 10 runs: 2684
No of sites: 1600 mean value for 10 runs: 6323
No of sites: 3200 mean value for 10 runs: 13287
No of sites: 6400 mean value for 10 runs: 26901
No of sites: 12800 mean value for 10 runs: 63254
No of sites: 25600 mean value for 10 runs: 143137
No of sites: 51200 mean value for 10 runs: 294382
```

Run the class for 10 times to system up and Plot these values on a graph:

| No of pairs generated | No of sites |
|---|---|
| 241 | 100 |
| 508 | 200 |
| 1313 | 400 |
| 2958 | 800 |
| 6090 | 1600 |




Take the log – log plot to see for a straight lune:




Now let us consider datasets and do math operations and visualize:

```
data = pd.read_excel("/Users/chara/Documents/DataSet_Assignment2.xlsx")
data
```

| | No of pairs generated | No of sites |
|---|---|---|
| **0** | 241 | 100 |
| **1** | 508 | 200 |
| **2** | 1313 | 400 |
| **3** | 2958 | 800 |
| **4** | 6090 | 1600 |

```
data['LogN'] = np.log(data['No of sites'])
data['LogM'] = np.log(data['No of pairs generated'])
data
```

| | No of pairs generated | No of sites | LogN | LogM |
|---|---|---|---|---|
| 0 | 241 | 100 | 4.605170 | 5.484797 |
| 1 | 508 | 200 | 5.298317 | 6.230481 |
| 2 | 1313 | 400 | 5.991465 | 7.180070 |
| 3 | 2958 | 800 | 6.684612 | 7.992269 |
| 4 | 6090 | 1600 | 7.377759 | 8.714403 |

```
data['LogN'] = np.log(data['No of sites (N)'])
#data['LogM'] = np.log(data['No of pairs generated (N)'])
data['NLogN'] = data['No of sites (N)'] * np.log(data['No of sites (N)'])
data
```

| | No of pairs generated (M) | No of sites (N) | LogN | NLogN |
|---|---|---|---|---|
| 0 | 241 | 100 | 4.605170 | 460.517019 |
| 1 | 508 | 200 | 5.298317 | 1059.663473 |
| 2 | 1313 | 400 | 5.991465 | 2396.585819 |
| 3 | 2958 | 800 | 6.684612 | 5347.689382 |
| 4 | 6090 | 1600 | 7.377759 | 11804.414253 |

```
data['LogN'] = np.log(data['No of sites (N)'])
#data['LogM'] = np.log(data['No of pairs generated (N)'])
data['1/2NLogN'] = 1/2*data['No of sites (N)'] * np.log(data['No of sites (N)'])
data
```

| | No of pairs generated (M) | No of sites (N) | LogN | 1/2NLogN |
|---|---|---|---|---|
| 0 | 241 | 100 | 4.605170 | 230.258509 |
| 1 | 508 | 200 | 5.298317 | 529.831737 |
| 2 | 1313 | 400 | 5.991465 | 1198.292909 |
| 3 | 2958 | 800 | 6.684612 | 2673.844691 |
| 4 | 6090 | 1600 | 7.377759 | 5902.207127 |

We have more similarities with M and 0.5NLogN.

We can conclude that the relation between no of pairs generated in random is similar to the ½*N*LogN.